GG

CERN_CN 90-16

03 JUIL. 1990

9

CERN-COMPUTING & NETWORKS DIVISION
CN/90/16
June 1990

CERN LIBRARIES, GENEVA

CM-P00065573

Q4 - A Simple Workload Manager for VM/XA

B. Pope

Q4 - A Simple Workload Manager for VM/XA

Bucky Pope

BUCKY at CERNVM

The VM/XA Operating System delivers cpu time in accordance with a userid's share designation. This is an efficient method for the immediate management of the system, but there are more global patterns of usage for a typical VM/XA system that fall outside the horizon of these mechanisms. These patterns relate to the daily cycle of user activity and the shifting between human intensive and compute intensive work. This paper describes a mechanism called Q4 which augments the scheduling and dispatching mechanisms of VM/XA. It is a relatively simple EXEC that identifies userids in a compute intensive stage and lowers their share. This effectively prevents users from overloading the system and assists in keeping the balance of resource delivery in line with installation policies. When userids have ceased operating in this intensive mode, Q4 returns them to a normal share that is better suited for interactive work.

Preface

The primary audience for this paper is VM/XA systems managers and systems programmers who are interested in workload management mechanisms. Most VM/XA installations are faced with the problem of balancing workload. Generally, batch or batch-like work has a lower priority for these systems. Many locations either have or want a mechanisms to help them enforce their policy for batch and interactive work.

Q4 was developed while on assignment at CERN. It was a rework of an EXEC called PERFORM developed by Sverre Jarp for the initial installation of VM/XA. As of the writing of this paper, the EXEC has been in production, in one of many forms, for over a year. It has been entirely rethought and rewritten at least twice, always in the hope to simplify and improve its effectiveness. My thanks to Sverre and the staff at CERN for letting me tinker with their system. The EXEC is also influenced by my experience at IBM Watson Research and the many years of experience with the NOW mechanism on VM/HPO. Although these two mechanisms are far from identical, they serve the same global function in the managing of VM systems.

The Myth of the Average User

This is no such thing as an average user. Figure 1 on page 2 shows the relationship of the percentage of usage of cpu time and the number of the users. This curve is so prevalent with individual computing systems that it can be considered universal. It is also the underlying problem with trying to manage a VM/XA system since the few users that consume most of the cpu time can destroy the performance of the remaining community. The Q4 EXEC is designed to help.

CERN, where the EXEC was developed, is a scientific installation for High Energy Physics (HEP) Research. The computing center supports thousands of scientists on various computers and operating systems. One of the major computers is an IBM 3090-600 running VM/XA. This system is essentially 100% busy twenty-four hours a day, seven days a week. Two thirds of the cpu time delivered is through their batch system. Users submit batch work into classes of different size jobs depending upon the total cpu time of the job. Batch work is released depending upon a daily schedule. Thus, the primary mechanism for delivery of cpu time is the batch system and it is CERN's policy that users should request cpu intensive work via the batch system.

During peak periods at CERN, there are as many as 1500 users logged on. Like any open system, it is difficult to enforce a policy such as CERN's since users have all the capabilities to run batch work interactively. In fact, this is a desirable property since interactive sessions are best for debugging and testing. However, this openness also leaves the opportunity for users to circumvent the batch policy by running work with their userid disconnected. It is CERN's policy to discourage this kind of activity, but not to prevent it. The problem then is to identify a user as a 'pseudo-batch' user, something that can only be determined by examining the user's behavior. Thus the need for a dynamic mechanism that watches for pseudo-batch users and alerts the system to deliver them cpu time at a rate lower than a real batch worker would receive. The EXEC described in this paper does just that. It is called Q4 because is operates beyond the sensitivity of the normal VM/XA Q1, Q2, and Q3 designations. It communicates with VM/XA by using the *Set Share* command. It also logs its activities for reporting purposes.

Static and Dynamic Policy

In order to have complete management of workload at an installation like CERN, one must consider two kinds of policies: static and dynamic. The static policy is one which is set and remains unchanged by the activities of the system and the workload. For example, CERN considers interactive workload more important than batch work. Therefore, they want the system scheduler to favor the interactive user over batch. This is done by setting different shares for the normal and batch users; i.e. normal is 100 and the batch users are 60. Interactive is also favored by limiting the number of batch workers. As long as the system is not 100% utilized by interactive work, the logged on users receive good response time. When the interactive work subsides during the night time and on weekends the service to batch work improves.

However, since people can run work interactively that they run in batch, there is a risk that they will consume available resources by substituting their userids for batch workers. If left unchecked, this behavior could undermine the service to other truly interactive users. Such a situation requires a dynamic workload management policy

and a mechanism to support it. The dynamic policy states that if any userid starts to behave like a batch worker, its service, as controlled by the share of the userid, must be no better than a batch worker. In fact, at CERN, the service for such a userid is made worse than that of a batch worker. The mechanism used to enforce this policy is Q4. At an installation like CERN with the intense demand for computing it is essential to have both types of policies.

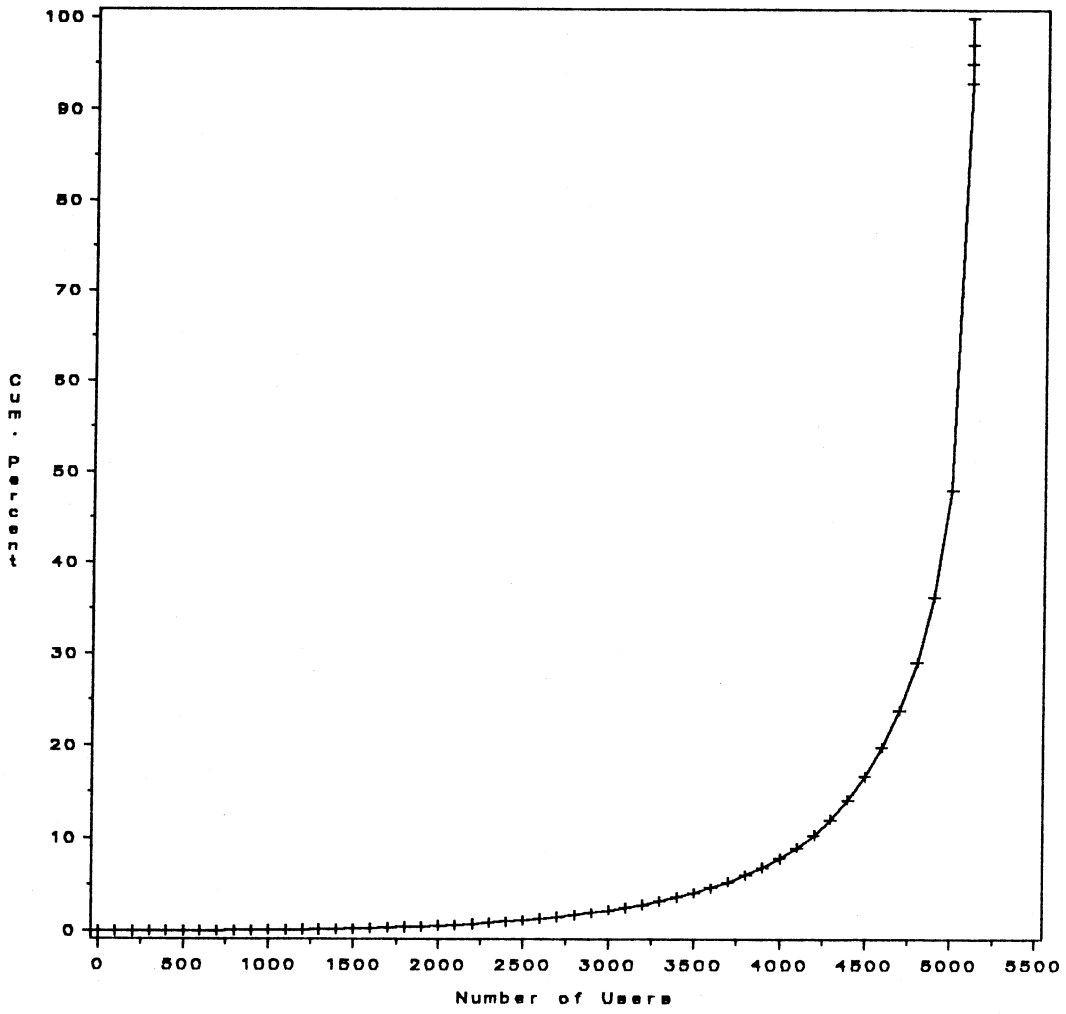

Figure 1. Distribution of CPU Time over CERN Users. The horizontal axis is the number of different active userids in the month of April 1990 on the CERN VM/XA system. The vertical axis is the cumulative percentage of cpu time. The users are organized from left to right by the amount of cpu time used. This curve demonstrates that the majority of cpu time is used by a small minority of the user community.

The remainder of this paper describes how Q4 operates, followed by a section showing some of the results and effects of Q4. Finally, in the appendix, is a copy of the current program.

The Q4 Exec

The objective was to keep Q4 simple and only use standard commands to avoid any reliance on special software or system modifications. Q4 must identify users that have started a long, compute intensive process in their userid. Such a process will quickly reach Q3 and stay there for a long time. My experience with VM has taught me that these are the users who 'hog' the cpu time. Some users may run short compute intensive work, but these users are not usually a problem since gaps between compute intensive segments averages their usage to a low level. In fact, it is desirable to allow users short compute intensive work with normal priority since this is very productive and satisfying for the users. It is longer term compute intensive work that creates the problem.

Given this assumption, I coded Q4 to use the CP INDCATE QUEUE command. It remembers all users in Q3. These are users who have already been computing for some time. At the same time, Q4 remembers the cpu time of this user taken from the CP INDICATE USER command. The next time Q4 is executed, it refers to its list of old Q3 users and calculates the cpu time used. The calculation works in percentage of a 3090 cpu, which is essentially the cpu time divided by the elapsed time.

At CERN, we execute Q4 every two minutes. We have also set the policy that no interactive user is allowed to consume more that 10% of a 3090 cpu for an extended period of time (i.e. two minutes). If a userid has consumed more than this, its share is lowered to 40 from the normal share of 100. The original share and the most recent cpu time is recorded in a file.

On the next iteration, a degraded userid's cpu usage is calculated again. If the userid has slowed its cpu consumption to less than 4% of a 3090 cpu, the userid's share is raised to 60. Then, on the next iteration, if the userid maintains this lower consumption rate, the share is returned to 100.

Not all userids are subject to this control. Userids that provide service to interactive users are exempt. Also, the batch userids all have their shares set to 60 in the directory, which keeps them below the interactive users. This is part of the static policy at CERN.

Selection of the Thresholds

Three numbers are very important to this algorithm: the elapsed time, the upper limit, and the reset limit. The elapsed time of two minutes was selected after some trial and error at CERN. We wanted to allow enough time for a user to perform a reasonable long cpu intensive task without penalty. Since the user can execute for one period without detection (it must be in Q3 to be detected), then for one more period before a measurement can be taken, it was felt by the CERN staff that a period of about two minutes would be reasonable. This allows time for a reasonable compile or script format.

The 10% of a cpu was initially an educated guess by the CERN staff. However, after some analysis of the performance data for the CERN system, I observed that the optimum performance, when the system was busy but the overhead was the least, was when there were 60 users in the dispatch list. Since this is a six way processor, there are 10 users per processor, allowing each userid to have 10%. In

fact, these numbers are approximate and not exact, but they indicated that 10% is about the right level.

The reset number of 4% was established by trial and error. Q4 logs each of its changes to share. One wants to avoid resetting a cpu intensive task until it's completed. We started with a reset of 6%, but found that frequently a userid would be reset to share 60 and then exceed the 10% usage again and be immediately reset to 40. After looking as some of the statistics, I tried the reset level of 4%, and now we rarely set users to share 60, then back to 40. It appears that on the CERN system, when a userid consumes less than 4% of a cpu, the compute intensive transaction is complete.

Results

Figure 2 illustrates the activity of Q4 changing the shares of users throughout the day. The greatest number of users whose shares are lowered to 40 occurs during the 3 P.M. interactive peak when there are 1500 userids logged on.

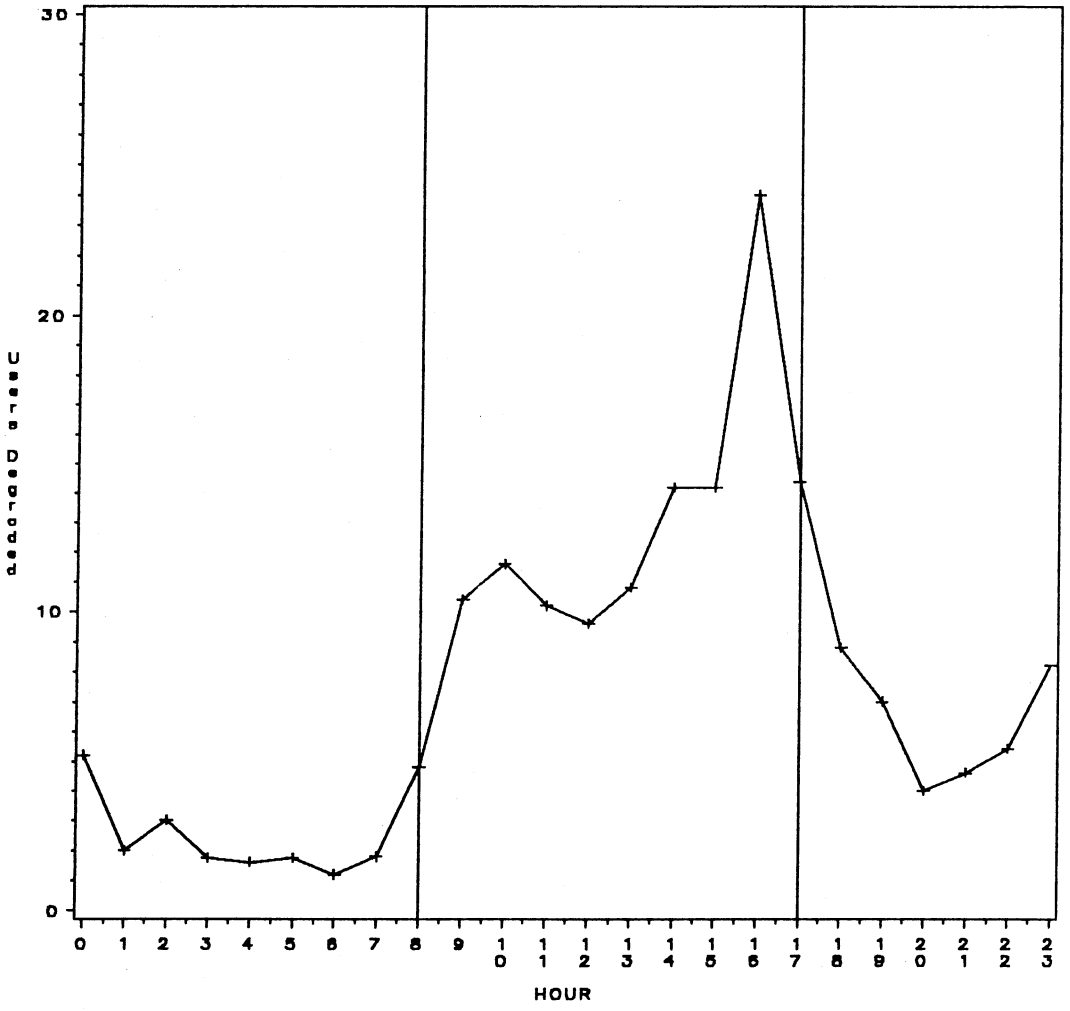

Figure 2. Frequency of Users Share changed by Q4. The horizontal axis shows the hours of the day. The vertical axis is the number of users whose shares are lowered to 40 each hour of the day.

The following table illustrates the effectiveness of the Q4 capability. The list shows, in descending order, the top 10 userids during the afternoon peak. Seven of them are batch jobs, none of which consume more than ten percent of one of the 3090's six cpus. Interactive users, those userids that do not begin with 'BATCH', are in the minority. Userids that demonstrated cpu intensive behavior are detected within two to four minutes and their share is reduced below the batch jobs. However, most interactive users, who are not cpu intensive, have their shares set at 100. They experience better priority than the batch jobs. The end result is a system that is fully utilized but still delivers a trivial response of less than .2 seconds.[1]

Userid	CPU %
MASLENNI	8.5
BATCH23	8.2
YAKA	7.4
BATCH27	7.2
BATCH32	6.8
DUGEAY	6.7
BATCH29	6.6
BATCH38	6.6
BATCH37	6.5
BATCH12	6.5

[1] Performance data for week days for the month of May, 1990, indicate that the CERN system was 99% utilized for the entire day, and the average prime shift trivial response was .16 seconds with a 90th percentile of .19.

Conclusion

In this paper, I document a capability on VM/XA called Q4. A number of installations have invented such a mechanism; it doesn't seem to be very complicated or expensive. The usual reason is to provide the VM scheduler with information about the relative importance of work. Usually, these locations must mix both interactive work and batch. It is desirable to have good performance for the interactive work, which can be done by keeping the batch work at a lower share. However, on VM/XA, there is nothing to prevent userids from running batch like work outside the batch system. Usually, this practice is discouraged, but not forbidden. The purpose of Q4 is to identify such behavior by a userid and lower its share below that of batch. In this way, the performance of interactive and batch work remains predictable while the pseudo batch work will receive the lowest priority of delivery of cpu time, in line with installation policy. A copy of the EXEC used at CERN is in the appendix. Hopefully, this will help a VM/XA site establish a successful workload management strategy of their own.

Appendix A. Q4 Exec

```
/*

Q4 - Bucky Pope (BUCKY @ CERNVM) September 1989

This EXEC is designed to operate in a VM/XA location that runs a
significant amount of batch work. Normally, BATCH workers are given a
priority lower than interactive users. However, some users will allow
long running batch like work to run under their userid. These are
called pseudo-batch workers and they can undermine the priority scheme
of a central computing installation.

Q4 is designed to identify pseudo-batch userids and lower their
share to bring them in line with BATCH priorities. Since BATCH jobs
are continuous, long running commands, they will remain in Q3 for
extended periods of time. Q4 must be executed on an interval
basis, say every 3 minutes. When it finds the same USERID in Q3 for
two successive periods, it calculates the % of a processor used. If
this exceeds an installation threshold (CPUPercentLimit) it will lower
the share of the offending userid (ShareBottomLimit). It will
continue to track this USERID until the usage drops below the
threshold (ResetPercent). It will then begin improving the SHARE
(ShareTopLimit). If usage remains down, the USERID is returned to
it's normal share.

NOTICE:

1) Q4 depends upon being called by some higher level process
   that controls the time intervals.

2) Q4 depends upon:
   a) Having adequate privilege to SET SHARE for userids (Class A)
   b) An exec called NONEXMPT that will eliminate userids that are
      exempt from having share changed.
   c) An exec called LOGHIST that will provide a LOG of the
      share changes.

3) Q4 keeps the history of users in a file called PAST
   Q3USERS. This file should not be changed or erased.

Reset Function:

Q4 RESET will reset all users to their original share.
No history is kept at this point.
*/

CpuPercentLimit   = .1
ResetPercent      = .04
NormalShare       = 100
ShareTopLimit     = 60
ShareBottomLimit  = 40
TimeThisIteration = time('S')
```

```
UsersToRemember  = ''
PastUserFile     = 'PAST' 'Q3USERS' 'A'
NotFound         = 0

if arg(1) = = '' then call ProcessQ3Users
else
  if arg(1) = '?' then
    do
      say 'Q4 manages share of Pseudo Batch userids.'
      say '       Read comments in the exec for more information.'
      exit
    end
  else if upper(arg(1)) = = 'RESET' then call ResetQ3Users
       else
         do
           say 'Parameter' arg(1) 'not recognized.'
           exit
         end

  LineOut = TimeThisIteration UsersToRemember
  address command 'EXECIO 1 DISKW' PastUserFile '1 (FINIS VAR LINEOUT'

exit

ProcessQ3Users:
  address command 'STATE' PastUserFile
  if rc = = 0 then call ProcessOldQ3Users

GetPresentQ3Users:
  PresentQ3Users = Q3Users()
  PresentQ3Users = RemoveExemptUsers(PresentQ3Users)

  do while PresentQ3Users < > ''
    parse var PresentQ3Users ThisUser PresentQ3Users
    if find(UsersToRemember,ThisUser) = = NotFound then
      do
        OriginalShare = qshare(ThisUser)
        if OriginalShare = = '' then iterate
        if OriginalShare < ShareBottomLimit then iterate
        if OriginalShare > NormalShare then
          do
            say 'More than Normal' thisuser originalshare
            iterate
          end
        PresentTtime  = ttime(ThisUser)
        call RememberThisUser
      end
  end

return

ResetQ3Users:
  address command 'STATE' PastUserFile
  if rc = = 0 then
    do
```

```
        address command 'EXECIO 1 DISKR' PastUserFile '(FINIS VAR LINEIN'
        parse var LineIn . LineIn
        do while LineIn < > ''
          parse var LineIn ThisUser OriginalShare OldTtime LineIn
          if ¬logged(ThisUser) then iterate
          PresentShare = qshare(ThisUser)
          if PresentShare = = '' then iterate
          if SomebodyElseChangedShare() then iterate
          if PresentShare = = OriginalShare then iterate
          call resetsharecommand OriginalShare
        end
      end

return

ProcessOldQ3Users:
        address command 'EXECIO 1 DISKR' PastUserFile '(FINIS VAR LINEIN'
        parse var LineIn TimeLastIteration LineIn
        if TimeThisIteration > TimeLastIteration then
          Duration = TimeThisIteration - TimeLastIteration
        else Duration = TimeThisIteration + (24*3600) - TimeLastIteration

        do while LineIn < > ''
          parse var LineIn ThisUser OriginalShare OldTtime LineIn
          if ¬logged(ThisUser) then iterate
          PresentShare = qshare(ThisUser)
          if PresentShare = = '' then iterate
           if SomebodyElseChangedShare() then
             do
              say 'Q4: Somebody changed' ThisUser,
                  'to' PresentShare
              iterate
             end
           PresentTtime = ttime(ThisUser)
           PercentUsed  = PercentUsed()

          Select

           when (PercentUsed > CpuPercentLimit) then
             if PresentShare = = ShareBottomLimit then call RememberThisUser
             else call setshare ShareBottomLimit

           when (PercentUsed > ResetPercent) then
             if PresentShare = = ShareTopLimit |,
                PresentShare = = ShareBottomLimit then
                 call RememberThisUser
             else call setshare min(OriginalShare,ShareTopLimit)

           otherwise

             Select

             when (PresentShare = = OriginalShare) then
                 iterate
```

```
            when (PresentShare = = ShareBottomLimit) then
                call setshare min(OriginalShare,ShareTopLimit)

            when (PresentShare = = ShareTopLimit) then
                call setshare OriginalShare

            otherwise iterate
            end
          end
        end
return

Q3Users: procedure
  InQueueList = cp('IND QUEUE')

  Q3List = ''

  do while InQueueList < > ''
    parse var InQueueList Userid Q . . InQueueList
    if Q = = 'Q3' then Q3List = Q3List Userid
  end

  return Q3List

PercentUsed:
      return  (CpuUsed() / Duration )

CpuUsed:
      if PresentTtime > = OldTtime then
        return PresentTtime-OldTtime
      else return PresentTtime

SomebodyElseChangedShare:
  if PresentShare = = OriginalShare |,
    PresentShare = = ShareTopLimit |,
    PresentShare = = ShareBottomLimit then return 0
  else return 1

RememberThisUser:
  if OriginalShare < > '' then
    if PresentTtime < > '' then
  UsersToRemember = UsersToRemember ThisUser OriginalShare PresentTtime
  return

logged: procedure
  parse upper value diagrc(8,'Q' arg(1)) with rc . . . LoggedMessage
  Loggedmessage = space(translate(LoggedMessage,' ','15'x))
  if LoggedMessage = 'NOT LOGGED ON' then return 0
  else return 1

SetShare:
  call RememberThisUser
ShareCommand:
  parse arg newshare
  if (datatype(newshare,'N') & datatype(newshare,'N')) then
```

```
    do
     'loghist' 'Share User' left(ThisUser,8),
        'NewShare' right(newshare,3),
        'OldShare' right(PresentShare,3),
        'Used' format(PercentUsed,,4)
     'CP SET SHARE' thisuser 'RELATIVE' newshare
    end
  else say 'One Share not Numbers: user' thisuser 'new' newshare,
         'old' oldshare
  return

ResetShareCommand:
  parse arg newshare
  if (datatype(newshare,'N') & datatype(newshare,'N')) then
    do
     'loghist' 'Reset User' left(ThisUser,8),
        'NewShare' right(newshare,3),
        'OldShare' right(PresentShare,3)
     'CP SET SHARE' thisuser 'RELATIVE' newshare
    end
  else say 'One Share not Numbers: user' thisuser 'new' newshare,
         'old' oldshare
  return

CP: procedure
  parse value diagrc(8,arg(1)) with rc . result
  if rc < > 0 then return ''
  else return translate(result,' ','15'x)

RemoveExemptUsers: return nonexmpt(arg(1))

ttime: procedure
  parse value cp('IND USER' arg(1)) with . 'TTIME=' min ':' sec .
  if min = = '' then return 0
  else return sec + (min*60)

qshare: procedure
  parse value cp('QUERY SHARE' arg(1)) ,
   with . ':' type . '=' share .
  if type = 'RELATIVE' then return share
  else return ''
```


Imprimé au CERN. Prix : 0.75 Fr.